Lab 01 – Java DataBase Connectivity (JDBC)

Neste laboratório faremos o uso da **API JDBC** para conectar ao **Banco de dados PostgreSQL** e executar instruções **SQL** como: **CREATE, INSERT, SELECT, UPDATE** e **DELETE**.

Utilizando o projeto realizado no modulo orientação a objetos, iremos modificar para que o mesmo utilize banco de dados, realizando as mesmas funções que o mesmo realizava.

Exercícios

Exercício 1: Instalar o **PostgreSQL** e criar o **Banco de Dados threeway**.

Exercício 2: Criar todas as tabelas referente ao projeto realizado no modulo de orientação a objetos.

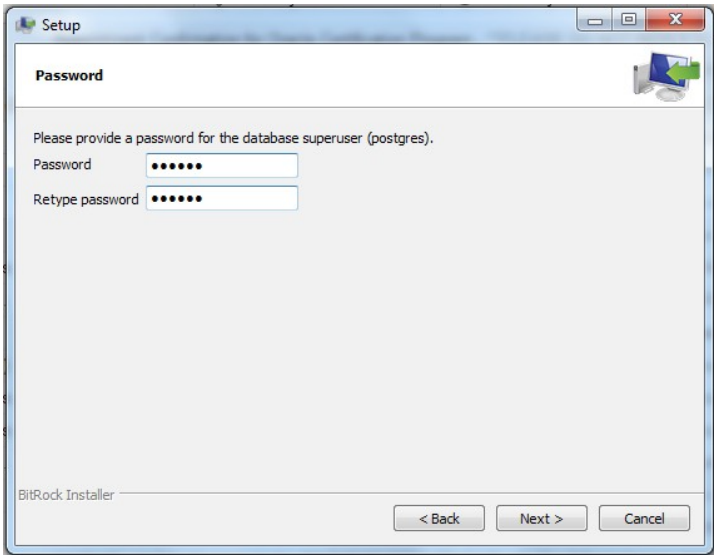
Exercício 3: Criar e Testar a classe **FabricaConexao.java**

Exercício 4: Iniciando o projeto **Livraria-Web**

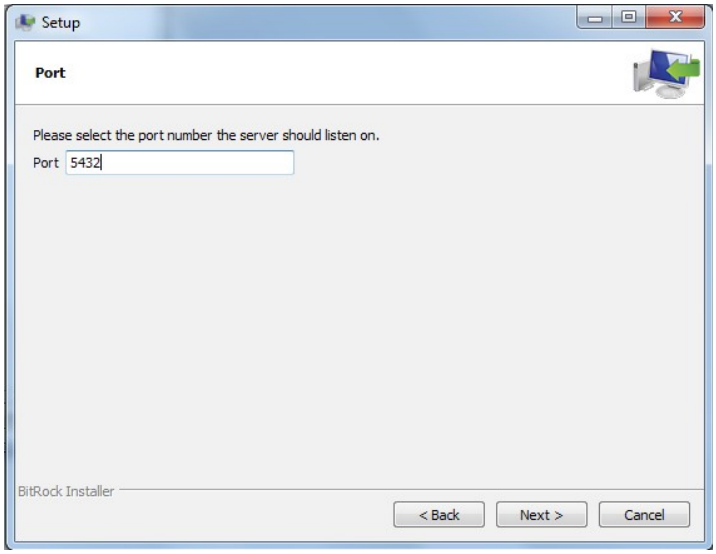
Exercício 1 – Instalar o PostgreSQL

1. Faça Download do instalador do postgreSQL no link: http://www.postgresql.org/download/.

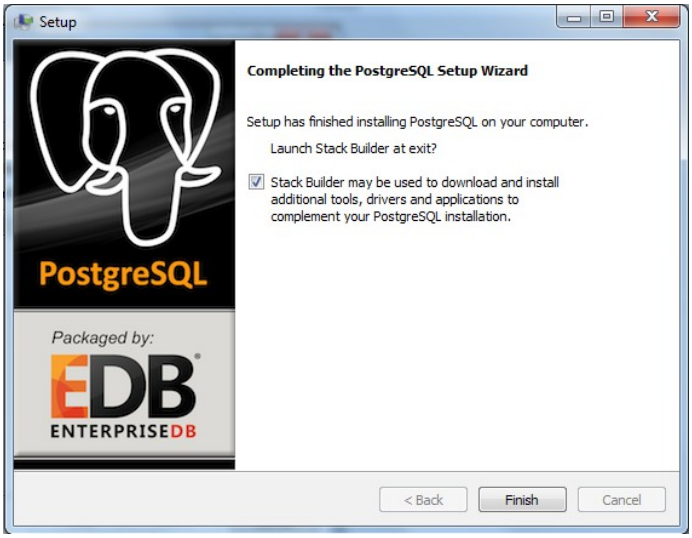
2. Execute o arquivo **postgresql-9.3.1-1-windows.msi**, selecione **Start** e depois selecione **NEXT** com as opções **Default** até a tela abaixo.



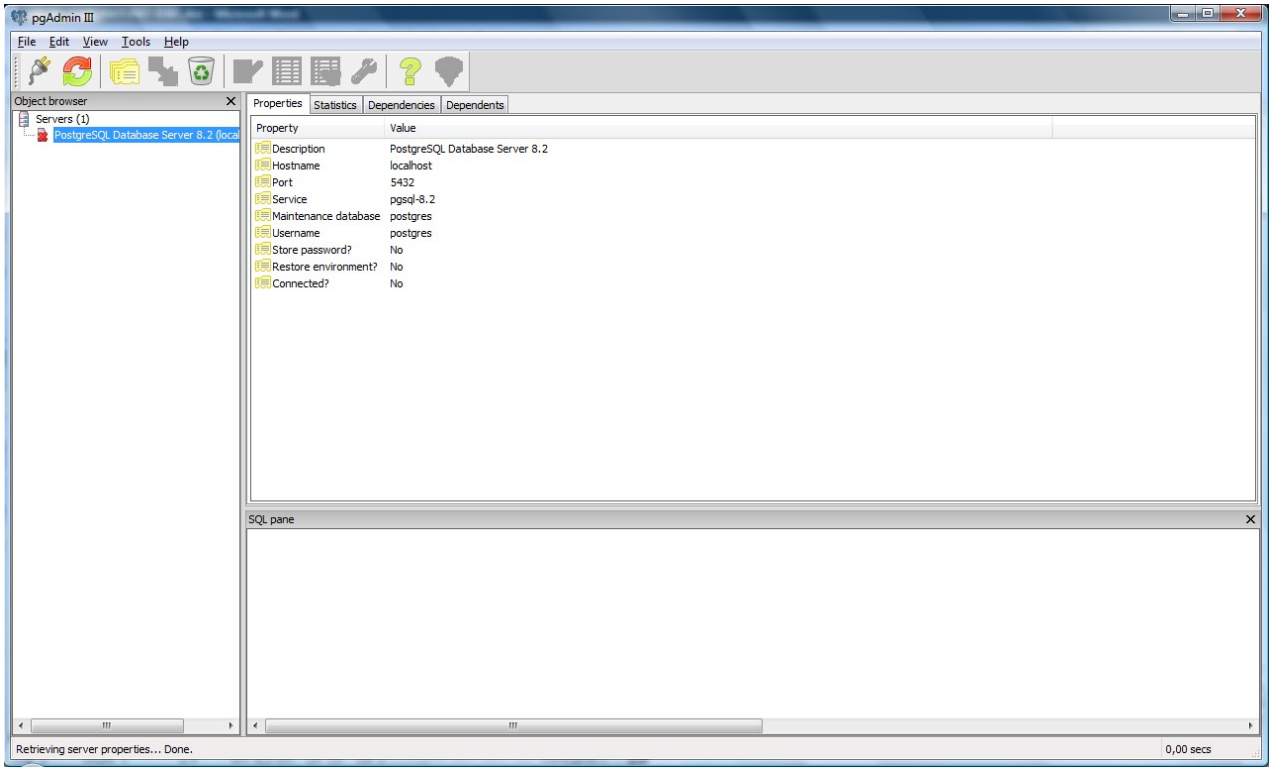
3. Informe a senha **123456**, para que todas as máquinas fiquem com a mesma senha de acesso e repita a senha **123456**, e selecione **Next**.



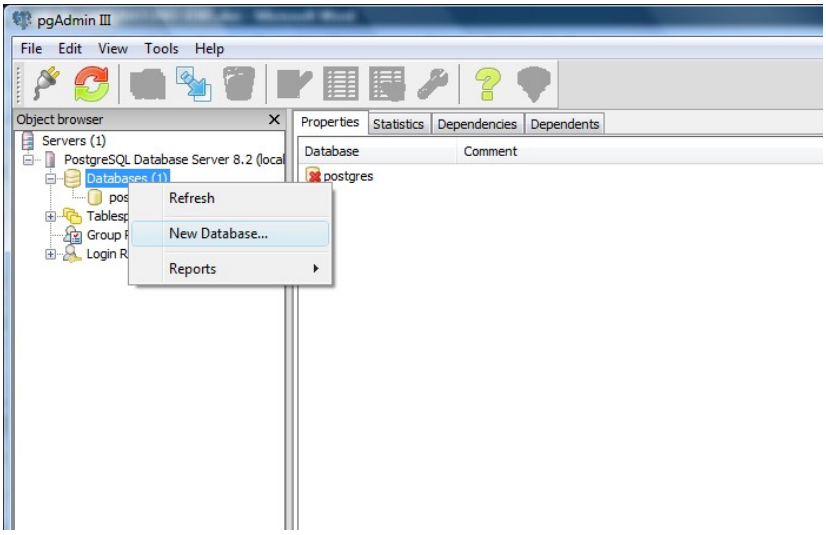
4. Confirme a porta do postgreSQL, a porta padrão e 5432

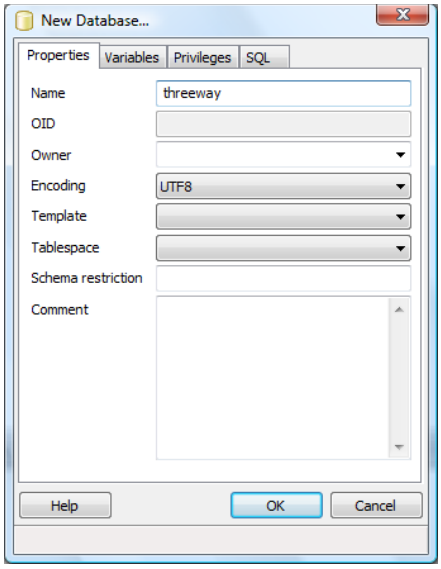


5. Após a instalação finalizar com sucesso, no menu **Iniciar,** selecione **pgAdmin III**. Esta é a ferramenta default de administração do banco de dados. Ao iniciá-la dê um duplo click na opção **PostgreSQL 9.3** para conectar ao banco de dados.



6. Clique com o botão direito do mouse em cima de **DataBases** e selecione a opção **NEW**, digite o nome **threeway**, selecione o **encoding UTF-8** e pressione **OK**. O banco será criado com toda estrutura necessária para criar as tabelas e acesso aos dados.





Exercício 2 – Criar todas as tabelas referente ao projeto realizado no modulo de orientação a objetos.

1. Agora vamos criar tabelas no Postgres de acordo com o que será usado no nosso projeto. Segue o sql da tabela abaixo:

```
CREATE TABLE cliente (
          COD_CLIENTE SERIAL NOT NULL,
          NOME VARCHAR(50) NOT NULL,
          LOGIN VARCHAR(10) NOT NULL,
          SENHA VARCHAR(10) NOT NULL,
          ENDERECO VARCHAR(32) NOT NULL,
          CIDADE VARCHAR(32) NOT NULL,
          BAIRRO VARCHAR(32) NOT NULL,
          ESTADO VARCHAR(32) NOT NULL,
          CEP VARCHAR(10) NOT NULL,
          PRIMARY KEY (COD_CLIENTE));

CREATE TABLE estoque (
          COD_LIVRO SERIAL NOT NULL,
          TITULO VARCHAR(30) NOT NULL,
          AUTOR VARCHAR(20) NOT NULL,
          PRECO NUMERIC NOT NULL,
          IMAGEM VARCHAR(80) NOT NULL,
          DESCRICAO VARCHAR(80),
          PRIMARY KEY (COD_LIVRO));

INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('GRANDE SERTAO - VEREDAS',
'ROSA, JOAO GUIMARAES', 165, 'imagens/veredas.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('QUANDO NIETZSCHE CHOROU',
'YALOM, IRVIN D.', 49.9, 'imagens/chorou.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('CASSINO ROYALE - JAMES
BOND 00', 'Fleming, Ian', 29.9, 'imagens/james.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('FILOSOFIA DO TEDIO',
'Svendsen, Lars', 29.9, 'imagens/tedio.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('O CASAMENTO', 'Rodrigues,
Nelson', 39.9, 'imagens/casamento.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('NEVE', 'PAMUK, ORHAN', 54,
'imagens/neve.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('VOLTA AO MUNDO EM OITENTA
DIAS', 'VERNE, JULIO', 16.5, 'imagens/volta_mundo.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('CRISTOVAO COLOMBO',
'VERNE, JULIO', 16.5, 'imagens/cristovao_colombo.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('VINTE MIL LEGUAS
SUBMARINAS', 'VERNE, JULIO', 14.9, 'imagens/submarinas.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('O SENHOR DOS ANEIS',
'TOLKIEN, J.R.R.', 169.9, 'imagens/senhor.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('HARRY POTTER', 'ROWLING,
J.K.', 89.7, 'imagens/harry.png');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('A AVENTURAS DE PI',
'MARTEL, YANN', 23.5, 'imagens/lifeofpi.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('PARA ONDE ELA FOI?',
'FORMAN, GAYLE', 20.0, 'imagens/onde.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('O LIVRO DO CEMITERIO',
'GAILMAN, NEIL', 20.0, 'imagens/cemiterio.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('SANDMAN VOL 1', 'GAILMAN,
NEIL', 489.0, 'imagens/sandman.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('WATCHMEN', 'MOORE, ALAN',
37.4, 'imagens/watchmen.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('JUSTICEIRO NOIR', 'TIIER,
```

```
FRANK', 12.5, 'imagens/justiceiro.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('SUPERMAN', 'TOMASI,
PETER', 5.9, 'imagens/superman.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('BATMAN', 'SNYDER, SCOTT',
5.9, 'imagens/batman.jpeg');


CREATE TABLE item_pedido (
          COD_ITEM SERIAL NOT NULL,
          QTD int NOT NULL,
          COD_LIVRO BIGINT NOT NULL,
          COD_PEDIDO BIGINT NOT NULL,
          PRIMARY KEY (COD_ITEM));


CREATE TABLE pedido (
          COD_PEDIDO SERIAL NOT NULL,
          DATA_PEDIDO DATE NOT NULL,
          STATUS VARCHAR(50) NOT NULL,
          COD_CLIENTE BIGINT NOT NULL,
          PRIMARY KEY (COD_PEDIDO));
```

Nesse SQL, estamos criando 4 tabelas, uma para Cliente, onde guardaremos as informações de usuários ; uma para estoque, onde serão inseridos os livros que ficarão disponíveis no site; uma para pedidos e outra para itens que compõem cada pedido. Perceba que a única tabela que está sendo preenchida agora é a de estoque. As outras serão preenchidas de acordo com o andamento do site.

Exercício 3 – Criar e testar a classe FabricaConexao

1. Crie o projeto java *Livraria-Servico* para configuração da classe de acesso ao banco de dados.

2. Vamos importar a biblioteca do driver **JDBC Postgres** para o projeto. Nossa aplicação precisa localizar o **Driver** de conexão com o banco na CLASSPATH do JVM. Como estamos usando o Eclipse, precisaremos adicionar a dependência do driver Postgres ao projeto. Faça download do driver no site https://jdbc.postgresql.org/download.html, baixe a versão compatível com a versão do seu JDK e do seu servidor de banco de dados PostgreSQL. Salve no o arquivo em um diretório com por exemplo **/home/usuario/java/libs .** No eclipse, selecione menu **Project → Properties → Java Build Path**, selecione a tab Libraries clique no botão **<<Add Externals JARs...>>**, selecione o arquivo da biblioteca do driver (*.jar, *.zip) e confirme botão **<<Ok>>**

3. Crie uma classe chamada **FabricaConexao.java** dentro do **pacote util** que tenha o método **estático getConexao()** que retorne uma conexão com o Banco de Dados.

```
import java.sql.*;

public class FabricaConexao {
static final String url = "jdbc:postgresql://localhost:5432/threeway";
static final String usuario = "postgres";
static final String senha = "123456";

public static Connection getConexao() throws SQLException {

        try{
                Class.forName("org.postgresql.Driver");
```

```
            return DriverManager.getConnection(url,usuario,senha);

        }catch(ClassNotFoundException e){
            throw new SQLException(e.getMessage());
        }
}
}
```

4. Crie um classe para testar a conexão com o banco de dados, dê o nome de **TestaConexao.java**.

```
import java.sql.*;

public class TestaConexao {

public static void main(String[] args) {
       Connection con;
       try {
            con = FabricaConexao.getConexao();
            if(con!=null)
                System.out.println("Conexao estabelecida!");

            con.close();

       } catch (SQLException e) {
            e.printStackTrace();
       }
}
}
```

5. Execute a classe **TestaConexao.java**, se ela imprimir no console **Conexao estabelecida!** siga em frente.

Exercício 4 – Iniciando o projeto Livraria-Web

4.1 – Classe Livro

1. A classe **Livro** representa a entidade livro em nosso projeto. Obedecerá o padrão de projeto VO (Value Object) e conterá os dados de um livro. Crie a classe **Livro** (dentro do pacote **model**) e adicione o seguinte código:

```
package model;

public class Livro {

      private int codigo;
      private String titulo;
      private String autor;
      private double preco;
      private String imagem;
      private String descricao;

      //getter e setters omitidos
}
```

2. Agora iremos criar a classe **LivroDao**. Essa classe será responsável pelo acesso ao banco de dados e retornar dados referentes a classe Livro. Crie a classe **LivroDao** (dentro do pacote **dao**) e adicione o seguinte código:

```
package dao;

import java.sql.*;

public class LivroDao {
    Logger LOG = Logger.getGlobal();

    private static final String OBTER_POR_ID_SQL = "SELECT AUTOR, TITULO, COD_LIVRO, IMAGEM,"
            + " PRECO, DESCRICAO FROM ESTOQUE WHERE COD_LIVRO = ?";

    private static final String CONSULTAR_SQL = "SELECT COD_LIVRO, TITULO, AUTOR, PRECO,"
            + " IMAGEM, DESCRICAO FROM ESTOQUE WHERE TITULO LIKE ?";

    public Livro consultar(int codigo) {

    public List<Livro> consultar(String titulo) {
    //Faça o restante do CRUD
}
```

Implemente os método consultar pelo código do livro, observe que estamos usando a tabela ESTOQUE, criada no exercício 2.

```
  public Livro consultar(int codigo) {
      Livro livro = null;
      try (Connection conexao = FabricaConexao.getConexao();
              PreparedStatement consulta = conexao.prepareStatement(OBTER_POR_ID_SQL);) {

          consulta.setInt(1, codigo);

          ResultSet resultado = consulta.executeQuery();

          if (resultado.next()) {
              livro = new Livro();
              livro.setAutor(resultado.getString("AUTOR"));
              livro.setCodigo(resultado.getInt("COD_LIVRO"));
              livro.setImagem(resultado.getString("IMAGEM"));
              livro.setPreco(resultado.getDouble("PRECO"));
              livro.setTitulo(resultado.getString("TITULO"));
              livro.setDescricao(resultado.getString("DESCRICAO"));
          }

          resultado.close();

      } catch (SQLException e) {
          LOG.severe(e.toString());
      }
      return livro;
  }
```

Implemente o método sobrecarregado consultar pelo titulo do livro, você de terminar a implementação, tome o método anterior como exemplo.

```
public List<Livro> consultar(String titulo) {
    ArrayList<Livro> lista = new ArrayList<Livro>();
    try (Connection conexao = FabricaConexao.getConexao();
            PreparedStatement consulta = conexao.prepareStatement(CONSULTAR_SQL);) {

        consulta.setString(1, "%" + titulo.toUpperCase() + "%");

        ResultSet resultado = consulta.executeQuery();

        while (resultado.next()) {
            Livro livro = new Livro();
            //COMPLETE VOCÊ O RESTANTE
            //DE CÓDIGO NECESSÁRIO
            lista.add(livro);
        }
        resultado.close();

    } catch (SQLException e) {
        LOG.severe(e.toString());
    }
    return lista;
}
```

3. Agora como teste, crie o arquivo **Pesquisa.java** e insira o código abaixo. Vamos percorrer o estoque de livros que foi adicionado no nosso banco de dados nos exercícios acima. Veja o resultado e entenda a estrutura de cada componente, alterando o nome do título pesquisado etc.

```
package dao.teste;
import java.util.Optional;
import dao.LivroDao;

public class Pesquisa {

     public static void main(String[] args) {

          String titulo = "CASSINO";

          LivroDao dao = new LivroDao();
          dao.consultar(titulo)
          .forEach(
               p -> System.out.println("Desc: "+ p.getDescricao() +"
               Preço:" + p.getPreco())
          );
     }

}
```